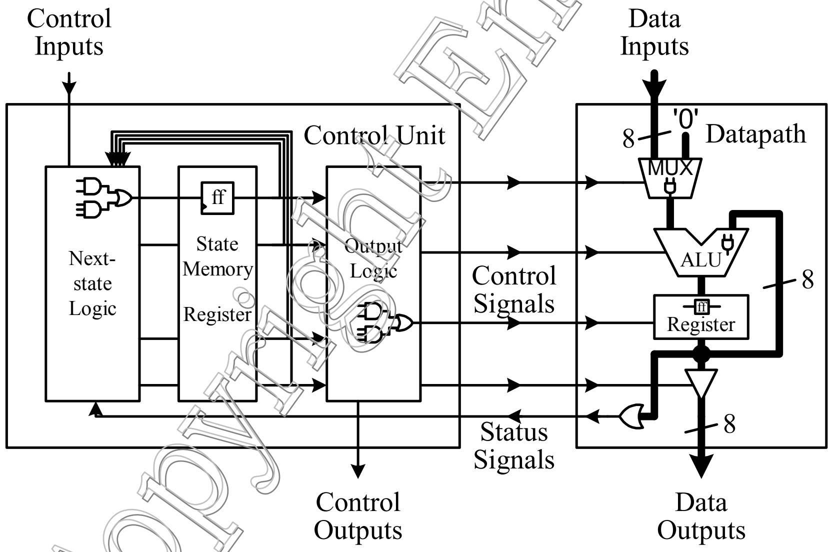
Contents

Chapter 5

Implementation Technologies



This chapter discusses how digital circuits are implemented at the physical level. As you know, transistors are the fundamental building blocks for all digital circuits. They are the actual physical devices that implement the binary switch and, therefore, also for the logic gates.

There are many different transistor technologies for creating a digital circuit. Some of these technologies are the diode-transistor logic (DTL), transistor-transistor logic (TTL), bipolar logic, and complementary metal-oxide-semiconductor (CMOS) logic. Among them, the most widely used is the CMOS technology.

Figure 5.1(a) shows a single discrete transistor with its three connections for signal input, output, and control. Above the transistor in the figure is a lump of silicon, which, of course is the main ingredient for the transistor. Figure 5.1(b) is a picture of transistors inside an IC taken with an electron microscope. Figure 5.1(c) is a higher magnification of the rectangle area in Figure 5.1(b).

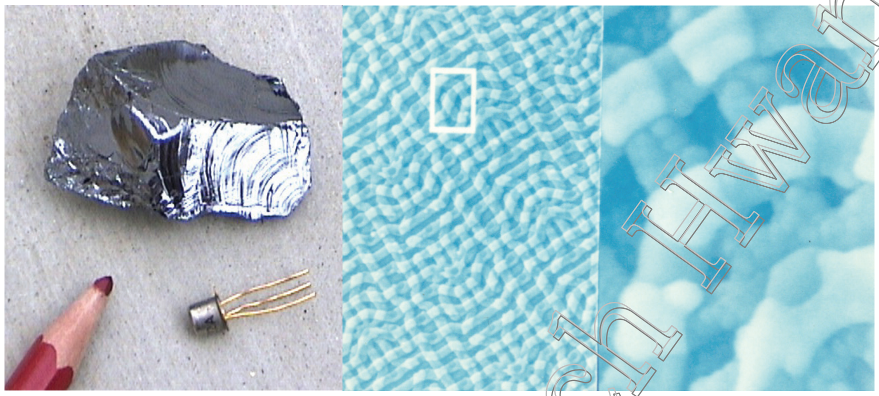

(a) (b) (c)

Figure 5.1 Transistors: (a) a lump of silicon and a transistor; (b) transistors inside an EPROM as seen through an electron microscope; (c) higher magnification of the rectangular area in (b).

In this chapter, we will look at how CMOS transistors work and how they are used to build the basic logic gates. Next, we will look at how digital circuits are actually implemented in various programmable logic devices (PLDs), such as read-only memories (ROMs), programmable logic arrays (PLAs), programmable array logic (PAL®) devices, complex programmable logic devices (CPLDs), and field programmable gate arrays (FPGAs). The optional Altera UP2 development board contains both a CPLD and a FPGA chip for implementing your circuits. The information presented in this chapter, however, is not needed to understand how microprocessors are designed at the logic circuit level.

5.1 Physical Abstraction

Physical circuits deal with physical properties, such as voltages and currents. Digital circuits use the abstractions 0 and 1 to represent the presence or absence of these physical properties. In fact, a range of voltages is interpreted as the logic 0, and another, non-overlapping range is interpreted as the logic 1. Traditionally, digital circuits operate with a 5-volt power supply. In such a case, it is customary to interpret the voltages in the range 0–1.5 V as logic 0, while voltages in the range 3.5–5 V as logic 1. This is shown in Figure 5.2. Voltages in the middle range (from 1.5–3.5 V) are undefined and should not occur in the circuit except during transitions from one state to the other. However, they may be interpreted as a "weak" logic 0 or a "weak" logic 1.

In our discussion of transistors, we will not get into their electrical characteristics of voltages and currents, but we will simply use the abstraction of 0 and 1 to describe their operations.

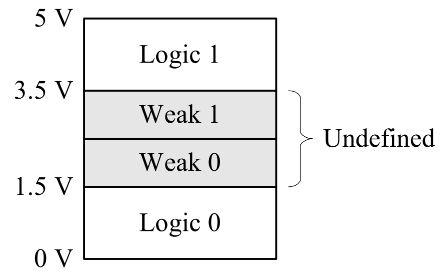


Figure 5.2 Voltage levels for logic 0 and 1.

5.2 Metal-Oxide-Semiconductor Field-Effect Transistor (MOSFET)

The metal-oxide-semiconductor field-effect transistor (MOSFET) acts as a voltage-controlled switch with three terminals: **source**, **drain**, and **gate**. The gate controls whether current can pass from the source to the drain or not. When the gate is asserted or activated, the transistor is turned on and current flows from the source to the drain. When looking at the transistor by itself, there is no physical difference between the source and the drain terminals. They are distinguished only when connected with the rest of the circuit by the differences in the voltage levels.

There are two variations of the MOSFET: the *n*-channel and the *p*-channel. The physical structures of these two transistors are shown in Figure 5.3(a) and (b), respectively. The name "metal-oxide-semiconductor" comes from the three layers of material that make up the transistor. The "*n*" stands for negative and represents the electrons, while "*p*" stands for positive and represents the holes that flow through a channel in the semiconductor material between the source and the drain.

For the *n*-channel MOSFET shown in Figure 5.3(a), a *p*-type silicon semiconductor material (called the substrate) is doped with *n*-type impurities at the two ends. These two *n*-type regions form the source and the drain of the transistor. An insulating oxide layer is laid on top of the two *n* regions and the *p* substrate, except for two openings leading to the two *n* regions. Finally, metal is laid in the two openings in the oxide to form connections to the source and the drain. Another deposit of metal is laid on top of the oxide between the source and the drain to form the connection to the gate.

The structure of the *p*-channel MOSFET shown in Figure 5.3(b) is similar to that in Figure 5.3(a), except that the substrate is of *n*-type material, and the doping for the source and drain is of *p*-type impurities.

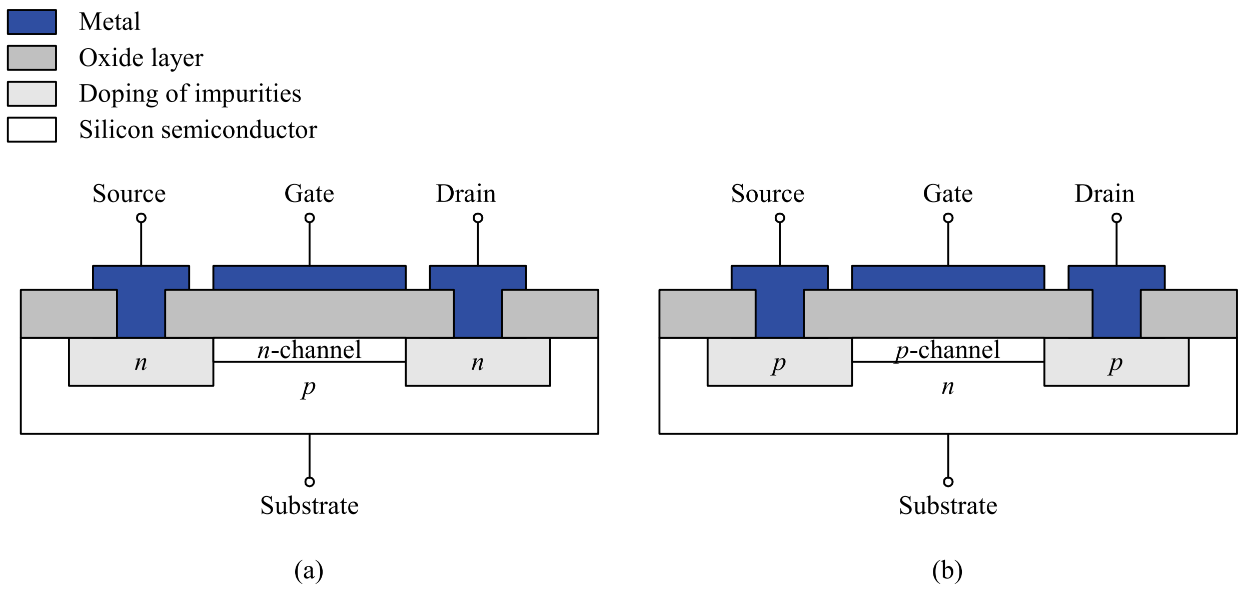


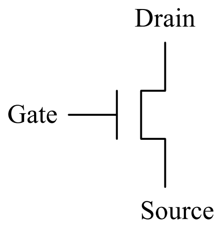
Figure 5.3 Physical structure of the MOSFET: (a) *n*-channel (NMOS); (b) *p*-channel (PMOS).

The *n*-channel and *p*-channel MOSFETs work opposite of each other. For the *n*-channel MOSFET, only an *n*-channel between the source and the drain is created under the control of the gate. This *n*-channel (*n* for negative) only allows negative charged electrons (0's) to move from the source to the drain. On the other hand, the *p*-channel MOSFET can only create a *p*-channel between the source and the drain under the control of the gate, and this *p*-channel (p for positive) only allows positive charged holes (1's) to move from the source to the drain.

5.3 CMOS Logic

In CMOS (complementary MOS) logic, only the two complementary MOSFET transistors, (*n*-channel also known as NMOS, and *p*-channel also known as PMOS)[1], are used to create the circuit. The logic symbols for the NMOS and PMOS transistors are shown in Figure 5.4(a) and Figure 5.5(a), respectively. In designing CMOS circuits, we are interested only in the three connections—source, drain, and gate—of the transistor. The substrate for the NMOS is always connected to ground, while the substrate for the PMOS is always connected to V_{CC}[2], so it is ignored in the diagrams for simplicity. Notice that the only difference between these two logic symbols is that one has a circle at the gate input, while the other does not. Using the convention that the circle denotes active-low (i.e., a 0 activates the signal), the NMOS gate input (with no circle) is, therefore, active-high. The PMOS gate input (with a circle) is active-low.

For the NMOS transistor, the source is the terminal with the lower voltage with respect to the drain. You intuitively can think of the source as the terminal that is supplying the 0 value, while the drain consumes the 0 value. The operation of the NMOS transistor is shown in Figure 5.4(b). When the gate is a 1 (asserted), the NMOS transistor is turned on or enabled, and the source input that is supplying the 0 can pass through to the drain output through the connecting *n*-channel. However, if the source has a 1, the 1 will not pass through to the drain even if the transistor is turned on, because the NMOS does not create a *p*-channel. Instead, only a weak 1 will pass through to the drain. On the other hand, when the gate is a 0 (or any value other than a 1), the transistor is turned off, and the connection between the source and the drain is disconnected. In this case, the drain will always have a high-impedance *Z* value independent of the source value. The × (don't-care) in the Input Signal column means it does not matter what the input value is, the output will be *Z*. The **high-impedance** value, denoted by ***Z***, means no value or no output. This is like having an insulator with an infinite resistance or a break in a wire, therefore whatever the input is, it will not pass over to the output.



(a)

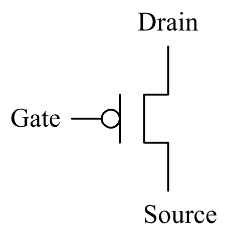
Gate	Switch	Input Signal	Output Signal
1	On (Closed)	0	0
		1	Weak 1
0 (Any value other than a 1)	Off (Open)	×	*Z*

(b)

Figure 5.4 NMOS transistor: (a) logic symbol; (b) truth table.

The PMOS transistor works exactly the opposite of the NMOS transistor. For the PMOS transistor, the source is the terminal with the higher voltage with respect to the drain. You intuitively can think of the source as the terminal that is supplying the 1 value, while the drain consumes the 1 value. The operation of the PMOS transistor is shown in Figure 5.5(b). When the gate is a 0 (asserted), the PMOS transistor is turned on or enabled, and the source input that is supplying the 1 can pass through to the drain output through the connecting *p*-channel. However, if the source has a 0, the 0 will not pass through to the drain even if the transistor is turned on, because the PMOS does not create an *n*-channel. Instead, only a weak 0 will pass through to the drain. On the other hand, when the gate is a 1 (or any value other than a 0), the transistor is turned off, and the connection between the source and the drain is disconnected. In this case, the drain will always have a high-impedance *Z* value independent of the source value.

[1] For bipolar transistors, these two transistors are referred to as NPN and PNP, respectively.
[2] V_{CC} is power or 5-volts in a 5 V circuit, while ground is 0 V.



Gate	Switch	Input Signal	Output Signal
0	On (Closed)	0	Weak 0
		1	1
1 (Any value other than a 0)	Off (Open)	×	*Z*

(a) (b)

Figure 5.5 PMOS transistor: (a) logic symbol; (b) truth table.

5.4 CMOS Circuits

CMOS circuits are built using only the NMOS and PMOS transistors. Because of the inherent properties of the NMOS and PMOS transistors, CMOS circuits are always built with two halves. One half will use one transistor type while the other half will use the other type, and when combined together to form the complete circuit, they will work in complement of each other. The NMOS transistor is used to output the 0 half of the truth table, while the PMOS transistor is used to output the 1 half of the truth table.

Furthermore, notice that the truth tables for these two transistors, shown in Figure 5.4(b) and Figure 5.5(b), suggest that CMOS circuits essentially must deal with five logic values instead of two. These five logic values are 0, 1, *Z* (high-impedance), weak 0, and weak 1. Therefore, when the two halves of a CMOS circuit are combined together, there is a possibility of mixing any combinations of these five logic values.

Figure 5.6 summarizes the result of combining these logic values. Here, a 1 combined with another 1 does not give you a 2, but rather, just a 1! A short circuit results from connecting a 0 directly to a 1 (that is, connecting ground directly to V_{CC}). This is like sticking two ends of a wire into the two holes of an electrical outlet in the wall. You know the result, and you don't want to do it! Connecting a 0 with a weak 1, or a 1 with a weak 0 will also result in a short, but it may take a longer time before you start to see smoke coming out. Any value combined with *Z* is just that value, since *Z* is nothing.

A properly designed CMOS circuit should always output either a 0 or a 1. Only the tri-state buffer also outputs the *Z* value. The other two values (weak 0 and weak 1) should not occur in any part of the circuit. The construction of several basic gates using the CMOS technology will now be shown.

	0	1	*Z*	Weak 0	Weak 1
0	0	Short	0	0	Short
1	Short	1	1	Short	1
Z	0	1	*Z*	Weak 0	Weak 1
Weak 0	0	Short	Weak 0	Weak 0	Short
Weak 1	Short	1	Weak 1	Short	Weak 1

Figure 5.6 Result of combining the five possible logic values.

5.4.1 CMOS Inverter

Half of the inverter truth table says that, given a 1, the circuit needs to output a 0. Therefore, the question to ask is which CMOS transistor (NMOS or PMOS) when given a 1 will output a 0? Looking at the two truth tables for the two transistors, we find that only the NMOS transistor outputs a 0. The PMOS transistor outputs either a 1 or a weak 0. A weak 0, as you recall from Section 5.1, is an undefined or an unwanted value. The next question to ask is how do we connect the NMOS transistor so that, when we input a 1, the transistor outputs a 0? The answer is shown in Figure 5.7(a) where the source of the NMOS transistor is connected to ground (to provide the 0 value), the gate is the input, and the drain is the output. When the gate is a 1, the 0 from the source will pass through to the drain output.

The complementary half of the inverter circuit is to output a 1 when given a 0. Again, from looking at the two transistor truth tables, we find that the PMOS transistor will do the job. This is expected, since we have used the NMOS for the first half, and the complementary second half of the circuit must use the other transistor. This time, the source is connected to V_{CC} to supply the 1 value, as shown in Figure 5.7(b). When the gate is a 0, the 1 from the source will pass through to the drain output.

To form the complete inverter circuit, we simply combine these two complementary halves together, as shown in Figure 5.7(c). When combining two halves of a CMOS circuit together, the one thing to be careful of is not to create any possible shorts in the circuit. We need to make sure that, for all possible combinations of 0's and 1's to all of the inputs, there are no places in the circuit where both a 0 and a 1 can occur at the same node at the same time.

For our CMOS inverter circuit, when the gate input is a 1, the bottom NMOS transistor is turned on while the top PMOS transistor is turned off. With this configuration, a 0 from ground will pass through the bottom NMOS transistor to the output, while the top PMOS transistor will output a high-impedance *Z* value. A *Z* combined with a 0 is still a 0, because a high impedance is of no value. Alternatively, when the gate input is a 0, the bottom NMOS transistor is turned off while the top PMOS transistor is turned on. In this case, a 1 from V_{CC} will pass through the top PMOS transistor to the output, while the bottom NMOS transistor will output a *Z*. The resulting output value is a 1. Since the gate input can never be both a 0 and a 1 at the same time, the output can only have either a 0 or a 1, and so, no short can result.

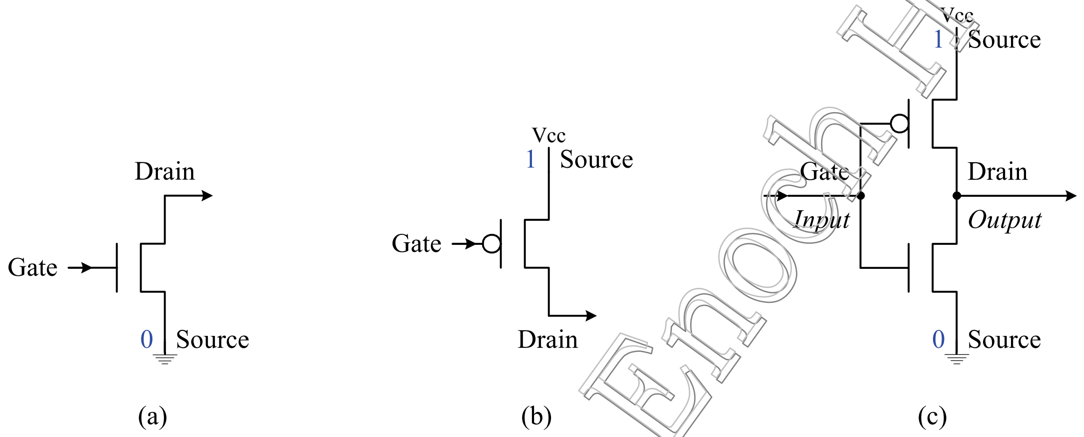


Figure 5.7 CMOS inverter circuit: (a) NMOS half; (b) PMOS half; (c) complete circuit.

5.4.2 CMOS NAND Gate

Figure 5.8 shows the truth table for the NAND gate. Half of the truth table consists of the one 0 output while the other half of the truth table consists of the three 1 outputs. For the 0 half of the truth table, we want the output to be a 0 when both *A* = 1 and *B* = 1. Again, we ask the question: Which CMOS transistor when given a 1 will output a 0? Of course, the answer is again the NMOS transistor. This time, however, since there are two inputs, *A* and *B*, we need two NMOS transistors. We need to connect these two transistors so that a 0 is output only when both are turned on with a 1. Recall from Section 2.3 that the AND operation results from two binary switches connected in series. Figure 5.9(a) shows the two NMOS transistors connected in series, with the source of one connected to ground to provide the 0 value, and the drain of the other providing the output 0. The two transistor gates are connected to the two inputs, *A* and *B*, so that only when both inputs are a 1 will the circuit output a 0.

The complementary half of the NAND gate is to output a 1 when either *A* = 0 or *B* = 0. This time, two PMOS transistors are used. To realize the OR operation, the two transistors are connected in parallel with both sources connected to V_{CC} and both drains to the output, as shown in Figure 5.9(b). This way, only one transistor needs to be turned on for the circuit to output the 1 value.

The complete NAND-gate circuit is obtained by combining these two halves together, as shown in Figure 5.9(c). When both *A* and *B* are 1, the two bottom NMOS transistors are turned on while the two top PMOS transistors are turned off. In this configuration, a 0 from ground will pass through the two bottom NMOS transistors to the output, while the two top PMOS transistors will output a high-impedance *Z* value. Combining a 0 with a *Z* will result in a 0.

Alternatively, when either *A* = 0 or *B* = 0 (or both equal to 0) at least one of the bottom NMOS transistors will be turned off, thus outputting a *Z*. On the other hand, at least one of the top PMOS transistors will be turned on and a 1 from V_{CC} will pass through that PMOS transistor. The resulting output value will be a 1. Again, we see that no short circuit can occur.

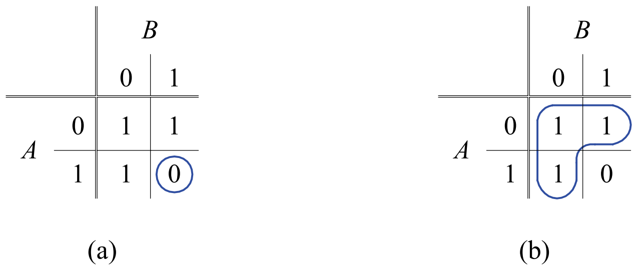


Figure 5.8 NAND-gate truth table: (a) the 0 half; (b) the 1 half.

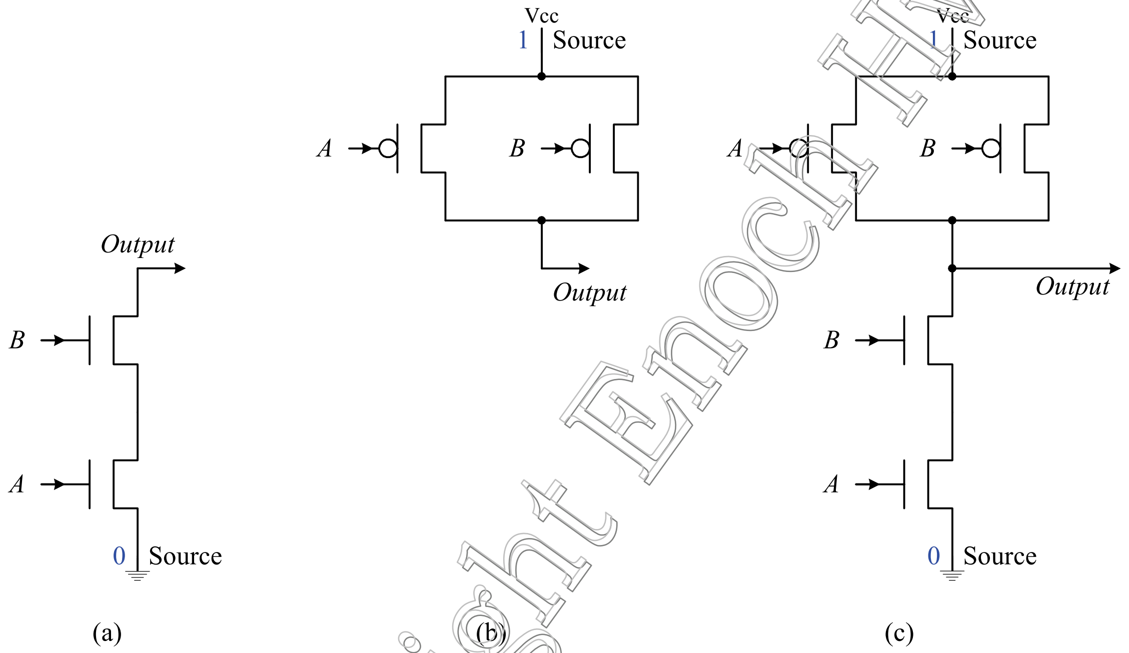


Figure 5.9 CMOS NAND circuit: (a) the 0 half using two NMOS transistors; (b) the 1 half using two PMOS transistors; (c) the complete NAND-gate circuit.

5.4.3 CMOS AND Gate

Figure 5.10 shows the 0 and 1 halves of the truth table for the AND gate. We can proceed to derive this circuit in the same manner as we did for the NAND gate. For the 0 half of the truth table, we want the output to be a 0 when either *A* = 0 or *B* = 0. This means that we need a transistor that outputs a 0 when it is turned on also with a 0. This, being one of the main differences between the NAND gate and the AND gate, causes a slight problem. Looking again at the two transistor truth tables in Figure 5.4 and Figure 5.5, we see that neither transistor fits this criterion. The NMOS transistor outputs a 0 when the gate is enabled with a 1, and the PMOS transistor outputs a 1 when the gate is enabled with a 0. If we pick the NMOS transistor, then we need to invert its input. On the other hand, if we pick the PMOS transistor, then we need to invert its output.

For this discussion, let us pick the PMOS transistor. To obtain the *A* or *B* operation, two PMOS transistors are connected in parallel. The output from these two transistors is inverted with a single NMOS transistor, as shown in Figure 5.11(a). When either *A* or *B* has a 0, that corresponding PMOS transistor is turned on, and a 1 from the V_{CC} source passes down to the gate of the NMOS transistor. With this NMOS transistor turned on, a 0 from ground is passed through to the drain output of the circuit.

For the 1 half of the circuit, we want the output to be a 1 when both A = 1 and B = 1. Again, we have the dilemma that neither transistor fits this criterion. To be complimentary with the 0 half, we will use two NMOS transistors connected in series. When both transistors are enabled with a 1, the output 0 needs to be inverted with a PMOS transistor, as shown in Figure 5.11(b).

Combining the two halves produces the complete AND-gate CMOS circuit shown in Figure 5.11(c). Instead of joining the two halves at the point of the output, the circuit connects together before inverting the signal to the output. The resulting AND-gate circuit is simply the circuit for the NAND gate followed by that of the INVERTER. From this discussion, we understand why (in practice) NAND gates are preferred over AND gates.

B
0 1
A
0 0 0
1 0 1
(a)
B
0 1
A
0 0 0
1 0 1
(b)

Figure 5.10 AND-gate truth table: (a) the 0 half; (b) the 1 half.

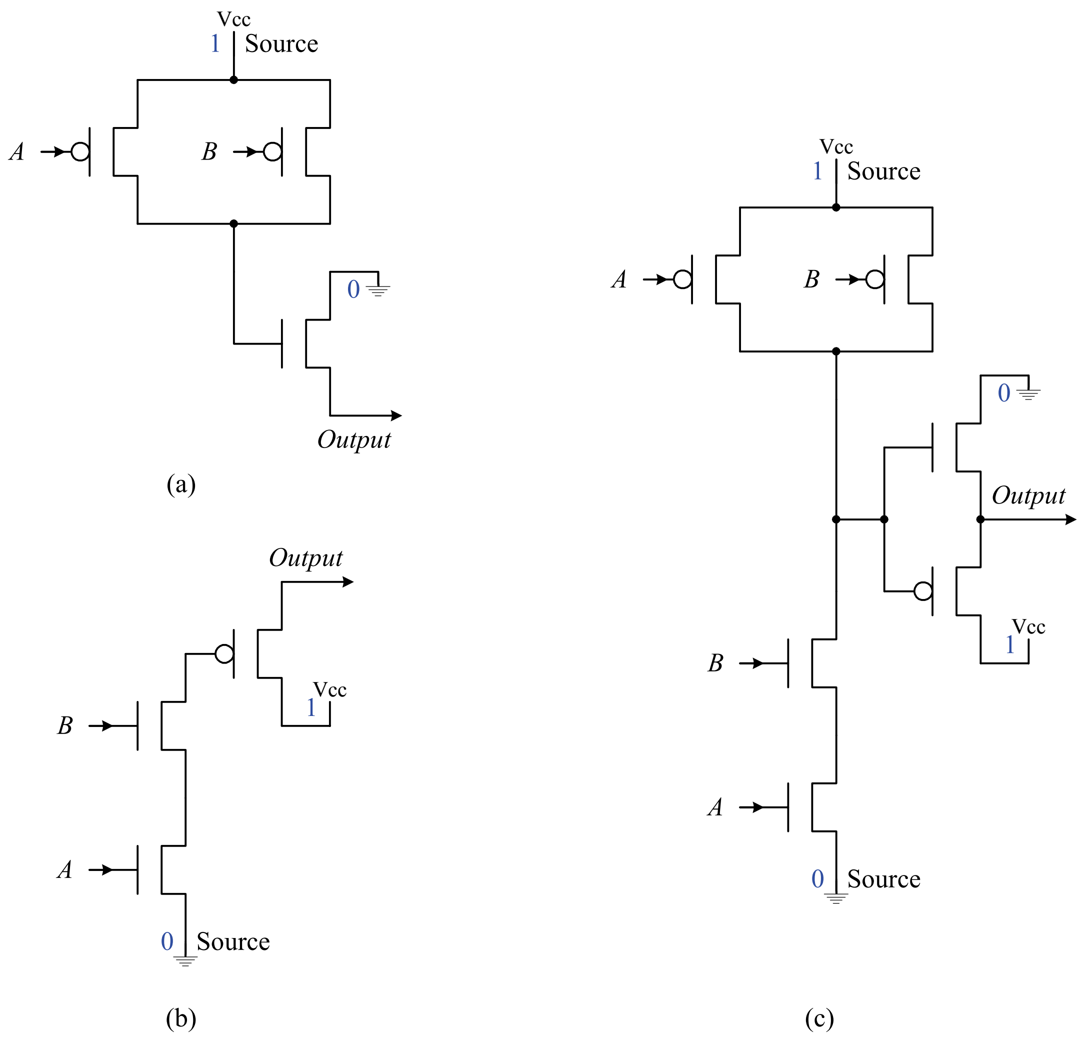


Figure 5.11 CMOS AND circuit: (a) the 0 half using two PMOS transistors and an NMOS transistor; (b) the 1 half

using two NMOS transistors and a PMOS transistor; (c) the complete AND-gate circuit.

5.4.4 CMOS NOR and OR Gates

The CMOS NOR-gate and OR-gate circuits can be derived similarly to that of the NAND- and AND- gate circuits. Like the NAND gate, the NOR-gate circuit uses four transistors, whereas the OR-gate circuit uses six transistors.

5.4.5 Transmission Gate

The NMOS and PMOS transistors, when used alone as a control switch, can pass only a 0 or a 1, respectively. Very often, we like a circuit that is able to pass both a 0 and a 1 under a control signal. The transmission gate is such a circuit that allows both a 0 and a 1 to pass through when it is enabled. When it is disabled, it outputs the *Z* value.

The transmission gate uses the two complimentary transistors connected together, as shown in Figure 5.12. Both ends of the two transistors are connected in common. The top PMOS transistor gate is connected to the inverted control signal, *C'*, while the bottom NMOS transistor gate is connected directly to the control signal, *C*. Hence, both transistors are enabled when the control signal *C* = 1, and the circuit is disabled when *C* = 0.

When the circuit is enabled, if the input is a 1, the 1 signal will pass through the top PMOS transistor, while the bottom NMOS transistor will pass through a weak 1. The final, combined output value will be a 1. On the other hand, if the input is a 0, the 0 signal will pass through the bottom NMOS transistor, while the top PMOS transistor will output a weak 0. The final, combined output value this time will be a 0. Therefore, in both cases, the output value is the same as the input value.

When the circuit is disabled with *C* = 0, both transistors will output the *Z* value. Thus, regardless of the input, there will be no output.

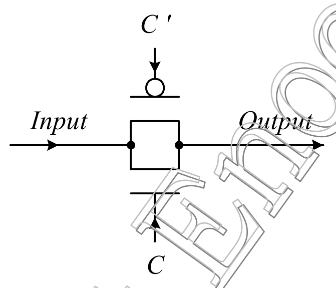


Figure 5.12 CMOS transmission-gate circuit.

5.4.6 2-input Multiplexer CMOS Circuit

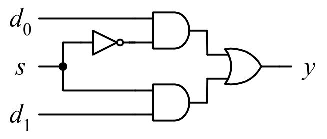
CMOS circuits for larger components can be derived by replacing each gate in the circuit with the corresponding CMOS circuit for that gate. Since we know the CMOS circuit for the three basic gates (AND, OR, and NOT) this is a simple "copy-and-paste" operation.

For example, we can replace the gate-level 2-input multiplexer circuit shown in Figure 5.13(a) with the CMOS circuit shown in Figure 5.13(b). For this circuit, we simply replace the two AND gates with the two 6-transistor circuits for the AND gate, another 6-transistor circuit for the OR gate, and the 2-transistor circuit for the INVERTER; giving a total of 20 transistors for this version of the 2-input multiplexer.

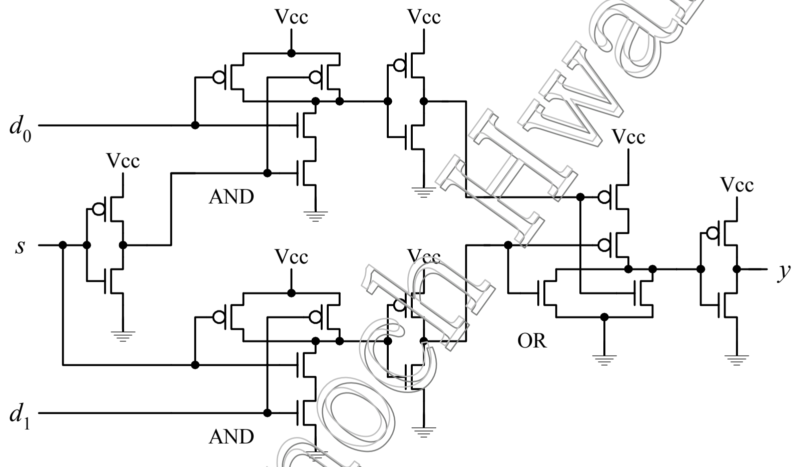
However, since the NAND gate uses two fewer transistors than the AND gate, we can convert the two-level OR-OF-ANDS circuit in Figure 5.13(a) to a two-level NAND-gate circuit shown in Figure 5.13(c). This conversion is based on the technology mapping technique discussed in Section 3.3. Performing the same "copy-and-paste" operation on this two-level NAND-gate circuit produces the CMOS circuit in Figure 5.13(d) that uses only 14 transistors.

We can do much better in terms of the number of transistors needed for the 2-input multiplexer circuit. From the original gate-level multiplexer circuit in Figure 5.13(a), we want to ask the question: What is the purpose of the two

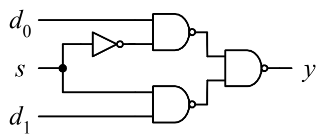
AND gates? The answer is that each AND gate acts like a control switch. When it is turned on by the select signal, *s*, the input passes through to the output. Well, the operation of the transmission gate is just like this, and it uses only two transistors. Hence, we can replace the two AND gates with two transmission gates. Furthermore, the AND gate outputs a 0 when it is disabled. In order for this 0 from the output of the disabled AND gate not to corrupt the data from the output of the other enabled AND gate, the OR gate is needed. If we connect the two outputs from the AND gates directly without the OR gate, a short circuit will occur when the enabled AND gate outputs a 1, because the disabled AND gate always outputs a 0. However, this problem disappears when we use two transmission gates instead of the two AND gates, because when a transmission gate is disabled, it outputs a Z value and not a 0. Thereby, we can connect the outputs of the two transmission gates directly without the need of the OR gate. The resulting circuit is shown in Figure 5.13(e), using only six transistors. The 2-transistor inverter is needed (just like in the gate-level circuit) for turning on only one switch while turning off the other switch at any one time.



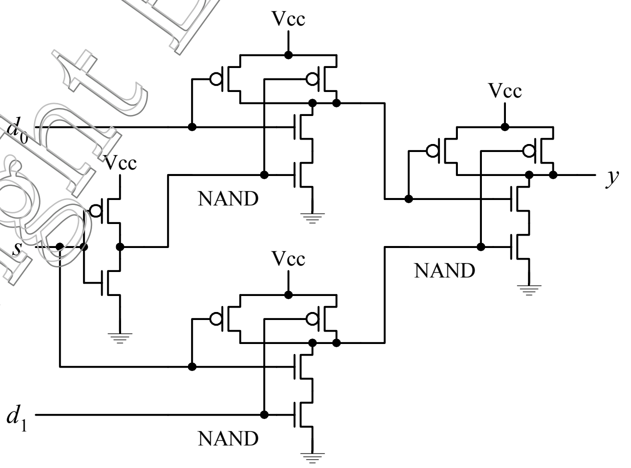
(a)



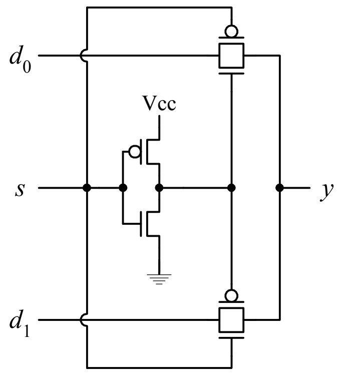
(b)



(c)



(d)

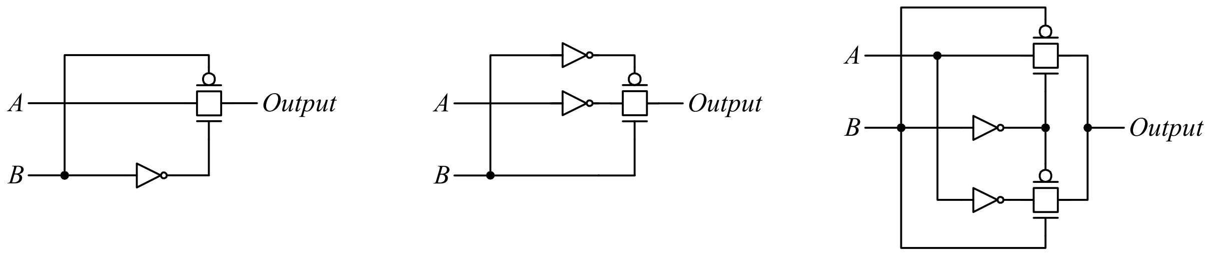


(e)

Figure 5.13 2-input multiplexer circuits: (a) gate-level circuit using AND and OR gates; (b) transistor-level circuit for part (a); (c) gate-level circuit using NAND gates; (d) transistor-level circuit for part (c); (e) transistor-level circuit using transmission gates.

5.4.7 CMOS XOR and XNOR Gates

The XOR circuit can be constructed using the same reasoning as for the 2-input multiplexer discussed in Section 5.4.6. First, we recall that the equation for the XOR gate is $AB' + A'B$. For the first AND term, we want to use a transmission gate to pass the A value. This transmission gate is enabled with the value B'. The resulting circuit for this first term is shown in Figure 5.14(a). For the second AND term, we want to use another transmission gate to pass the A' value and have the transmission gate enabled with the value B, resulting in the circuit shown in Figure 5.14(b). Combining the two partial circuits together gives us the complete XOR circuit shown in Figure 5.14 (c). Again, as with the 2-input multiplexer circuit, it is not necessary to use an OR gate to connect the outputs of the two transmission gates together.



(a) (b) (c)

Figure 5.14 CMOS XOR gate circuit: (a) partial circuit for the term AB'; (b) partial circuit for the term $A'B$; (c) complete circuit.

The CMOS XOR circuit shown in Figure 5.14(c) uses eight transistors: four transistors for the two transmission gates and another four transistors for the two inverters. However, with some ingenuity, we can construct the XOR circuit with only six transistors, as shown in Figure 5.15(a). Similarly, the XNOR circuit is shown in Figure 5.15(b). In the next section, we will perform an analysis of this XOR circuit to see that it indeed has the same functionality as the XOR gate.

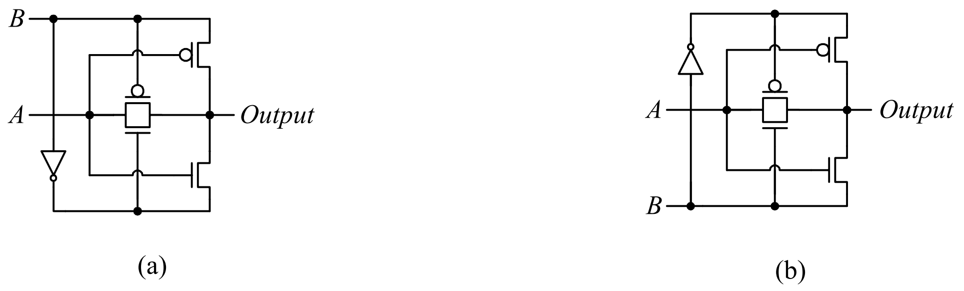


Figure 5.15 CMOS circuits using only six transistors for: (a) XOR gate; (b) XNOR gate.

5.5 Analysis of CMOS Circuits

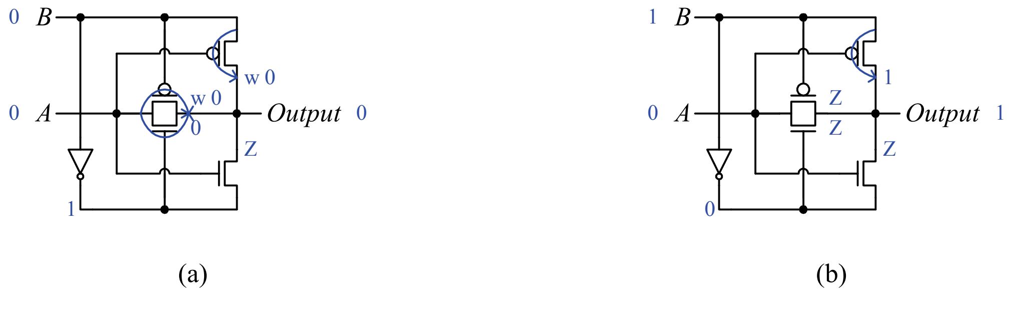
The analysis of a CMOS circuit follows the same procedure as with the analysis of a combinational circuit, as discussed in Section 3.1. First, we must assume that the inputs to the circuit must have either a logic 0 or logic 1 value; that is, the input value cannot be a weak 0, a weak 1, or a *Z*. Then, for every combination of 0 and 1 to the inputs, trace through the circuit (based on the operations of the two CMOS transistors) to determine the value obtained at every node in the circuit. When two different values are merged together at the same point in the circuit, we will use the table in Figure 5.6 to determine the resulting value.

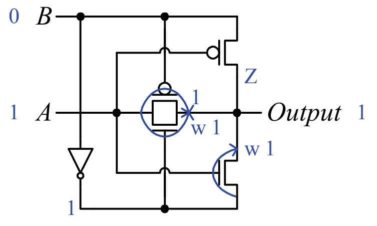
Example 5.1: Analyzing the XOR CMOS circuit

Analyze the CMOS circuit shown in Figure 5.15. For this discussion, the words "top-right," "top-middle," "bottom-middle," and "bottom-right" are used to refer to the four transistors in the circuit.

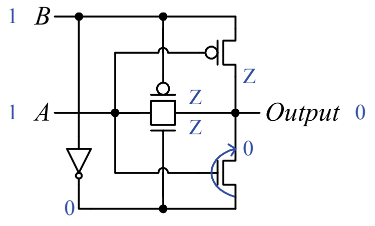
Figure 5.16(a) shows the analysis of the circuit with the inputs *A* = 0 and *B* = 0. The top-right PMOS transistor is enabled with a 0 from input *A*; however, the source for this transistor is a 0 from input *B*, and this produces a weak 0 at the output of this transistor. In Figure 5.16, the arrow denotes that the transistor is enabled, and the label "w 0" at its output denotes that the output value is a weak 0. For the top-middle PMOS transistor, it is also enabled but with a 0 from input *B*. The source for this transistor is a 0 from *A*, and so, the output is again a weak 0. The bottom-middle NMOS transistor is enabled with a 1 from *B'*. Since the source is a 0 from *A*, this transistor outputs a 0. For the bottom-right NMOS transistor, the 0 from *A* disables it, and so, a Z value appears at its output. The outputs of these four transistors are joined together at the point of the circuit output. At this common point, two weak 0's, a 0, and a Z are combined together. Referring to Figure 5.6, combining these four values together results in an overall value of a 0. Hence, the circuit outputs a 0 for the input combination *A* = 0 and *B* = 0.

Figure 5.16(b), (c), and (d) show the analysis of the circuit for the remaining three input combinations. The outputs for all four input combinations match exactly those of the 2-input XOR gate. ♦





(c)

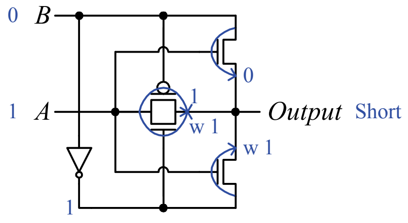


(d)

Figure 5.16. Analysis of the CMOS XOR-gate circuit: (a) shows the analysis for the inputs *AB* = 00. All the transistor outputs are annotated with the resulting output value. The letter "w" is used to signify that it is a weak value; (b) through (d) show the analysis for the remaining input combinations 01, 10, and 11, respectively.

Example 5.2: Analyzing a CMOS circuit with a short

The CMOS circuit shown below is modified slightly from the XOR circuit from Example 5.1; the top-right PMOS transistor is replaced with a NMOS transistor. Let us perform an analysis of this circuit using the inputs *A* = 1 and *B* = 0.

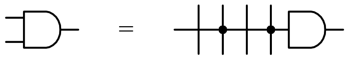


The top-right NMOS transistor is enabled with a 1 from input *A*. The source for this NMOS transistor is a 0 from input *B*, and so, it outputs a 0. The top-middle PMOS transistor is also enabled, but with a 0 from input *B*. The source for this PMOS transistor is a 1 from input *A*, and so, it outputs a 1. It is not necessary to continue with the analysis of the remaining two transistors, because at the common output, we already have a 0 (from the top-right NMOS transistor) combining with a 1 (from the top-middle PMOS transistor) producing a short circuit. ♦

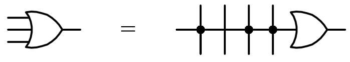
5.6 * Using ROMs to Implement a Function

Memory is used for storing binary data. This stored data, however, can be interpreted as being the implementation of a combinational circuit. A combinational circuit expressed as a Boolean function in canonical form is implemented in the memory by storing data bits in appropriate memory locations. Any type of memory, such as ROM (read-only memory), RAM (random access memory), PROM (programmable ROM), EPROM (erasable PROM), EEPROM (electrically erasable PROM), and so on, can be used to implement combinational circuits. Of course, non-volatile memory is preferred, since you do want your circuit to stay intact after the power is removed.

In order to understand how combinational circuits are implemented in ROMs, we need to first understand the internal circuitry of the ROM. ROM circuit diagrams are drawn more concisely by the use of a new logic symbol to represent a logic gate. Figure 5.17 shows this new logic symbol for an AND gate and an OR gate with multiple inputs. Instead of having multiple input lines drawn to the gate, the input lines are replaced with just one line going to the gate. The multiple input lines are drawn perpendicular to this one line. To actually connect an input line to the gate, an explicit connection point (•) must be drawn where the two lines cross. For example, in Figure 5.17(a), the AND gate has only two inputs; whereas in (b), the OR gate has three inputs.

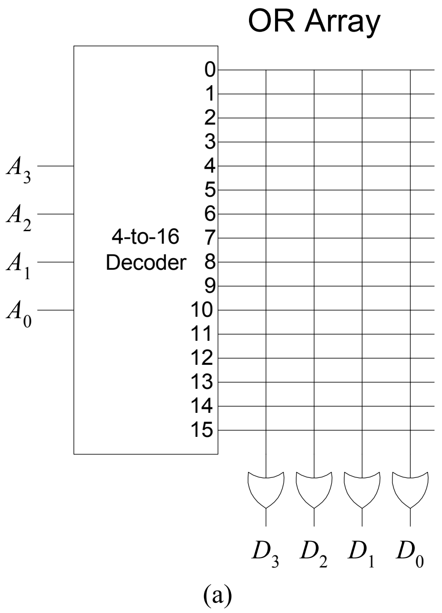


(a)



(b)

Figure 5.17 Array logic symbol for: (a) AND gate; (b) OR gate.



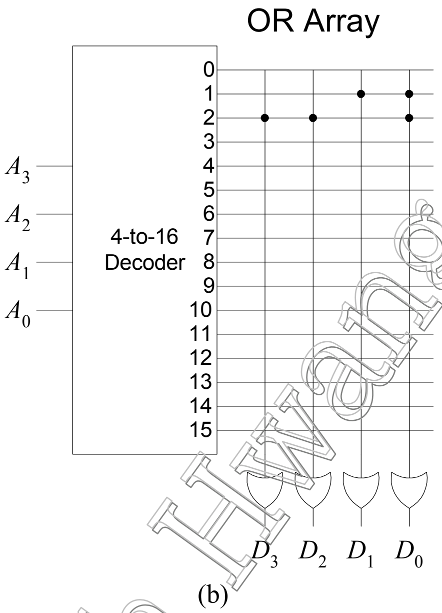


Figure 5.18 Internal circuit for a 16 × 4 ROM: (a) with no connections made; (b) with connections made.

The circuit diagram for a 16 × 4 ROM having 16 locations, each being 4-bits wide, is shown in Figure 5.18(a). A 4-to-16 decoder is used to decode the four address lines, A_3, A_2, A_1, and A_0, to the 16 unique locations. Each output of the decoder is a location in the memory. Recall that the decoder operation is such that, when a certain address is presented, the output having the index of the binary address value will have a 1, while the rest of the outputs will have a 0.

Four OR gates provide the four bits of data output for each memory location. The area for making the connections between the outputs of the decoder with the inputs of the OR gates is referred to as the OR array. When no connections are made, the OR gates always will output a 0, regardless of the address input. With connections made as in Figure 5.18(b), the data output of the OR gates depends on the address selected. For the circuit in Figure 5.18(b), if the address input is 0000, then the decoder output line 0 will have a 1. Since there are no connections made between the decoder output line 0 and any of the four OR gate inputs, the four OR gates will output a 0. Therefore, the data stored in location 0 is 0000 in binary. If the address input is 0001, then the decoder output line 1 will have a 1. Since this line is connected to the inputs of the two OR gates for D_1 and D_0, therefore, D_1 and D_0 will both have a 1, while D_3 and D_2 will both have a 0. Hence, the data stored in location 1 is 0011. In the circuit of Figure 5.18(b), the value stored in location 2 is 1101.

A 16 × 4 ROM can be used to implement a 4-variable Boolean function as follows. The four variables are the inputs to the four address lines of the ROM. The 16 decoded locations become the 16 possible minterms for the 4-variable function. For each 1-minterm in the function, we make a connection between that corresponding decoder output line that matches that minterm number with the input of an OR gate. It does not matter which OR gate is used, as long as one OR gate is used to implement one function. Hence, up to four functions with a total of four variables can be implemented in a 16 × 4 ROM, such as the one shown in Figure 5.18(a). Larger sized ROMs, of course, can implement larger and more functions.

From Figure 5.18(b), we can conclude that the function associated with the OR gate output, D_0, is $F = \Sigma(1,2)$. That is, minterms 1 and 2 are the 1-minterms for this function while the rest of the minterms are the 0-minterms. Similarly, the function for D_1 has only minterm 1 as its 1-minterm. The functions for D_2 and D_3 both have only minterm 2 as its 1-minterm.

ROMs are programmed during the manufacturing process and cannot be programmed afterwards. As a result, using ROMs to implement a function is only cost effective if a large enough quantity is needed. For small quantities, EPROMs or EEPROMs are preferred. Both EPROMs and EEPROMs can be programmed individually using an inexpensive programmer connected to the computer. The memory device is inserted into the programmer. The bits to be stored in each location of the memory device are generated by the development software. This data file is then transferred to the programmer, which then actually writes the bits into the memory device. Furthermore, both EPROMs and EEPROMs can be erased and reprogrammed with different data bits.

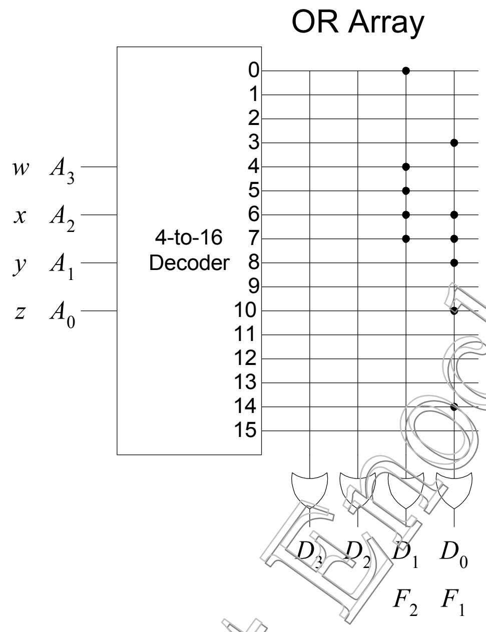
Example 5.3: Using a 16 × 4 ROM to implement Boolean functions

Implement the following two Boolean functions using the 16 × 4 ROM circuit shown in Figure 5.18.

$F_1(w,x,y,z) = w'x'yz + w'xyz' + w'xyz + wx'y'z' + wx'yz' + wxyz'$

$F_2(w,x,y,z) = w'x'y'z' + w'x$

For F_1, the 1-minterms are m_3, m_6, m_7, m_8, m_{10}, and m_{14}. For F_2, the 1-minterms are m_0, m_4, m_5, m_6, and m_7. Notice that in F_2, the term $w'x$ expands out to four minterms. The implementation is shown in the circuit connection below. We arbitrarily pick D_0 to implement F_1 and D_1 to implement F_2.



♦

5.7 * Using PLAs to Implement a Function

Using ROMs or EPROMs to implement a combinational circuit is very wasteful because usually many locations in the ROM are not used. Each storage location in a ROM represents a minterm. In practice, only a small number of these minterms are the 1-minterms for the function being implemented. As a result, the ROM implementing the function usually is quite empty.

Programmable logic arrays (**PLAs**) are designed to reduce this waste by not having all of the minterms "built-in" as in ROMs but rather, by allowing the user to specify only the minterms that are needed. PLAs are designed specifically for implementing combinational circuits.

The internal circuit for a 4 × 8 × 4 PLA is shown in Figure 5.19. The main difference between the PLA circuit and the ROM circuit is that, in the PLA circuit, an AND array is used instead of a decoder. The input signals are available both in the inverted and non-inverted forms. The AND array allows the user to specify only the product terms needed by the function; namely, the 1-minterms. The OR array portion of the circuit is similar to that of the ROM, allowing the user to specify which product terms to sum together. Having four OR gates will allow up to four functions to be implemented in a single device.

In addition, the PLA has an output array which provides the capability to either invert or not invert the value at the output of the OR gate. This is accomplished by connecting one input of the XOR gate to either a 0 or a 1. By connecting one input of the XOR gate to a 1, the output of the XOR gate is the inverse of the other input. Alternatively, connecting one input of the XOR gate to a 0, the output of the XOR gate is the same as the other input. This last feature allows the implementation of the inverse of a function in the AND/OR arrays, and then finally getting the function by inverting it.

The actual implementation of a combinational circuit into a PLA device is similar to writing data bits into a ROM or other memory device. A PLA programmer connected to a computer is used. The development software allows the combinational circuit to be defined and then transferred and programmed into the PLA device.

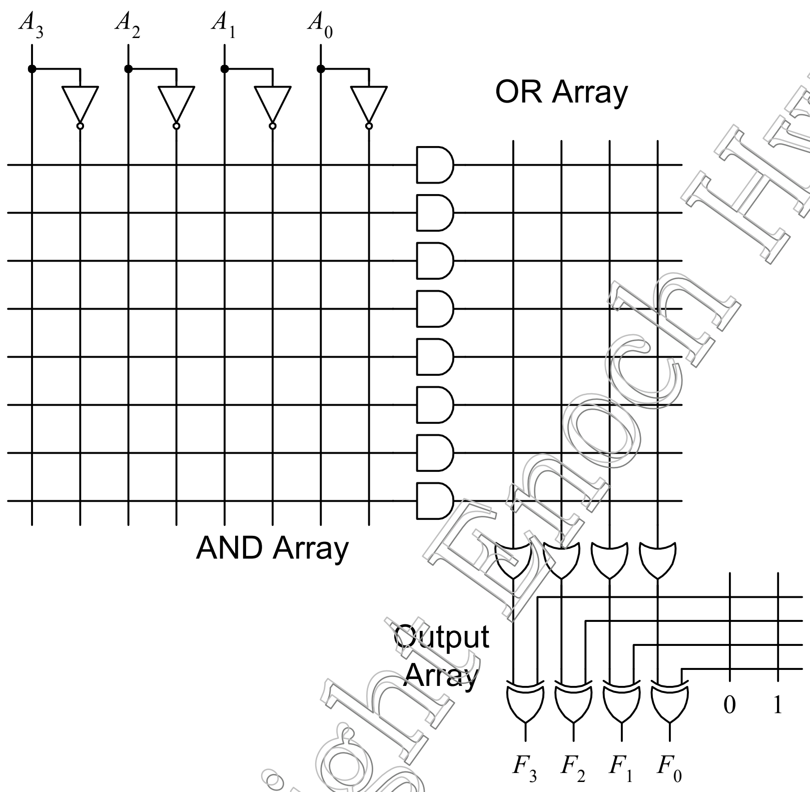


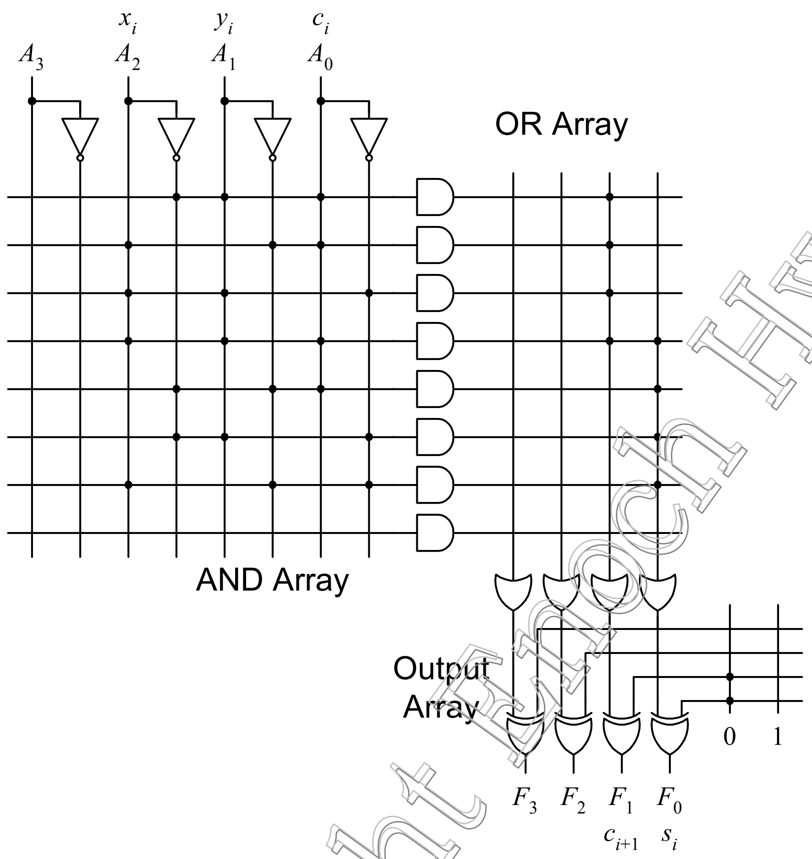
Figure 5.19 Internal circuit for a 4 × 8 × 4 PLA.

Example 5.4: Using a 4 × 8 × 4 PLA to implement a full adder circuit

Implement the full adder circuit in a 4 × 8 × 4 PLA. The truth table for the full adder from Section 4.2.1 is shown here.

x_i	y_i	c_i	c_{i+1}	s_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

In the PLA circuit shown next, the three inputs, x_i, y_i, and c_i, are connected to the PLA inputs, A_2, A_1, and A_0, respectively. The first four rows of the AND array implement the four 1-minterms of the function c_{i+1}, while the next three rows of the AND array implement the first three 1-minterms of the function s_i. The last minterm, m_7, is shared by both functions, and therefore, it does not need to be duplicated. The two functions, c_{i+1} and s_i, are mapped to the PLA outputs, F_1 and F_0, respectively. Since the two functions are implemented directly (i.e., not the inverse of the functions), the XOR gates for both functions are connected to 0.



♦

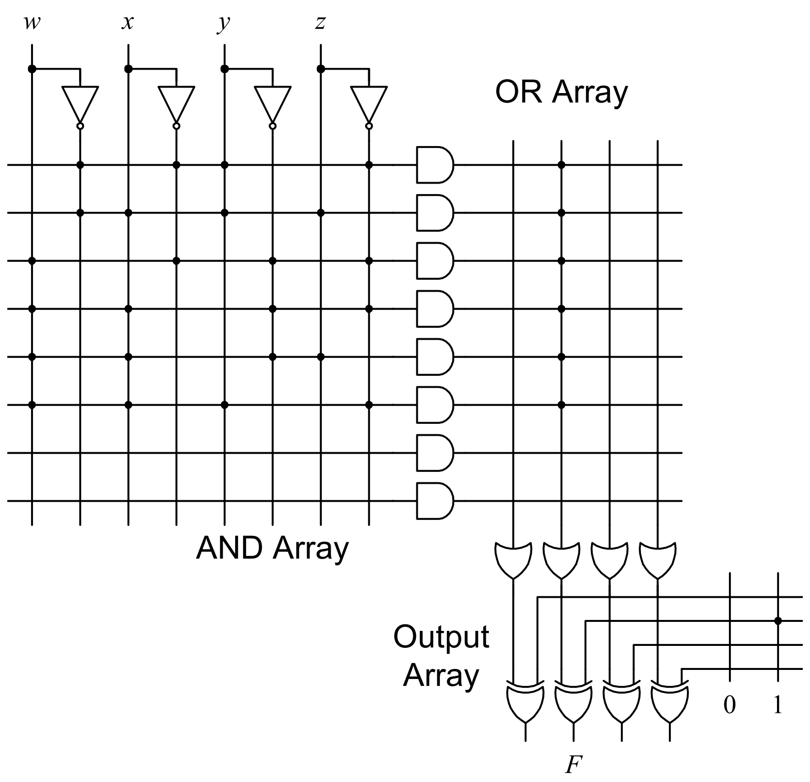
Example 5.5: Using a 4 × 8 × 4 PLA to implement a function

Implement the following function in a 4 × 8 × 4 PLA.

$F(w,x,y,z) = \Sigma(0, 1, 3, 4, 5, 6, 9, 10, 11, 15)$

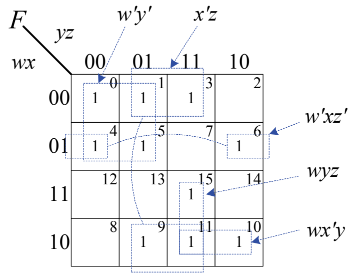
This four-variable function has ten 1-minterms. Since the 4 × 8 × 4 PLA can accommodate only eight minterms, we need to implement the inverse of the function, which will have only six 1-minterms (16 – 10 = 6). The inverse of the function can then be inverted back to the original function at the output array by connecting one of the XOR inputs to a 1, as shown here.

$$\begin{aligned} F' &= \Sigma(2, 7, 8, 12, 13, 14) \\ &= w'x'yz' + w'xyz + wx'y'z' + wxy'z' + wxy'z + wxyz' \end{aligned}$$

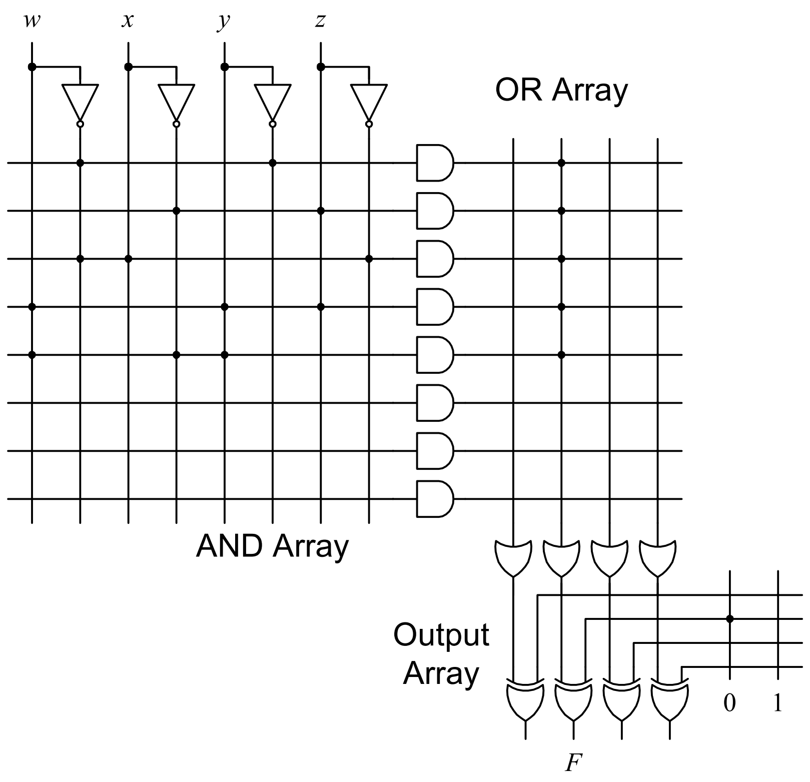


Another way to implement the above function in the PLA is to first minimize it. The following K-map shows that the function reduces to

F = *w'y'* + *x'z* + *w'xz'* + *wyz* + *wx'y*



With only five product terms, the function can be implemented directly without having to be inverted, as shown in the following circuit.



♦

5.8 * Using PALs to Implement a Function

Programmable array logic (**PAL**®) devices are similar to PLAs, except that the OR array for the PAL is not programmable but rather, fixed by the internal circuitry. Hence, they are not as flexible in terms of implementing a combinational circuit.

The internal circuit for a 4 × 4 PAL is shown in Figure 5.20. The OR gate inputs are fixed; whereas, the AND gate inputs are programmable. Each output section is from the OR of the three product terms. This means that each function can have, at most, three product terms. To make the device a little bit more flexible, the output F_3, is fed back to the programmable inputs of the AND gates. With this connection, up to five product terms are possible for one function.

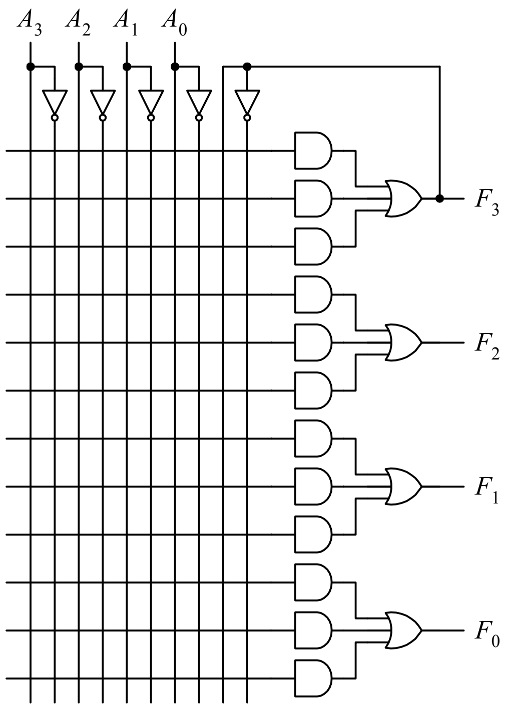


Figure 5.20. Internal circuit for a 4 × 4 PAL device.

Example 5.6: Using a 4 × 4 PAL to implement functions

Implement the following three functions given in sum-of-minterms format using the PAL circuit of Figure 5.20.

$$F_1(w,x,y,z) = w'x'yz + wx'yz'$$

$$F_2(w,x,y,z) = w'x'yz + wx'yz' + w'xy'z' + wxyz$$

$$F_3(w,x,y,z) = w'x'y'z' + w'x'y'z + w'x'yz' + w'x'yz$$

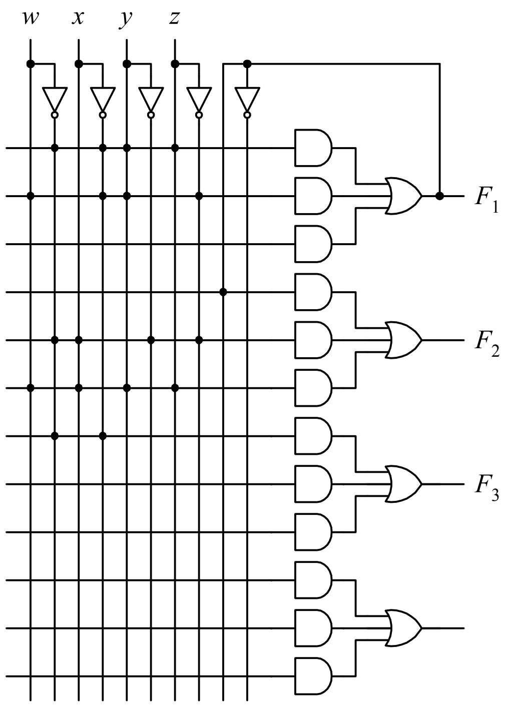
Function F_1 has two product terms, and it can be implemented directly in one PAL section. F_2 has four product terms, and so, it cannot be implemented directly. However, we note that the first two product terms are the same as F_1. Hence, by using F_1, it is possible to reduce F_2 from four product terms to three as shown here.

$$\begin{aligned} F_2(w,x,y,z) &= w'x'yz + wx'yz' + w'xy'z' + wxyz \\ &= F_1 + w'xy'z' + wxyz \end{aligned}$$

F_3 also has four product terms, but these four product terms can be reduced to just one by minimizing the equation as shown here.

$$\begin{aligned} F_3(w,x,y,z) &= w'x'y'z' + w'x'y'z + w'x'yz' + w'x'yz \\ &= w'x'(y'z' + y'z + yz' + yz) \\ &= w'x' \end{aligned}$$

The connections for these three functions are shown in the following PAL circuit. Notice that, for functions F_1 and F_3, there are unused AND gates. Since there are no inputs connected to them, they output a 0, which does not affect the output of the OR gate.



♦

5.9 * Complex Programmable Logic Device (CPLD)

Using ROMs, PLAs, and PALs to implement a combinational circuit is fairly straightforward and easy to do. However, to implement a sequential circuit or a more complex combinational circuit may require more sophisticated and larger programming devices. The **complex programmable logic device** (**CPLD**) is capable of implementing a circuit with upwards of 10,000 logic gates.

The CPLD contains many PAL-like blocks that are connected together using a programmable interconnect to form a matrix. The PAL-like blocks in the CPLD are called *macrocells*, as shown in Figure 5.21. Each macrocell has a programmable-AND-fixed-OR array similar to a PAL device for implementing combinational logic operations. The XOR gate in the macrocell circuit, shown in Figure 5.21, will either invert or not invert the output from the combinational logic. Furthermore, a flip-flop (discussed in Chapter 6) is included to provide the capability of implementing sequential logic operations. The flip-flop can be bypassed using the multiplexer for combinational logic operations.

Groups of 16 macrocells are connected together to form the logic-array blocks. Multiple logic-array blocks are linked together using the programmable interconnect, as shown in Figure 5.22. Logic signals are routed between the logic-array blocks on the programmable interconnect. This global bus is a programmable path that can connect any signal source to any destination on the CPLD.

The input/output (I/O) blocks allow each I/O pin to be configured individually for input, output, or bi-directional operation. All I/O pins have a tri-state buffer that is controlled individually. The I/O pin is configured as an input port if the tri-state buffer is disabled; otherwise, it is an output port.

Figure 5.23 shows some of the main features of the Altera MAX7000 CPLD. Instead of needing a separate programmer to program the CPLD, all MAX devices support *in-system programmability* through the IEEE JTAG interface. This allows designers to program the CPLD after it is mounted on a printed circuit board. Furthermore, the device can be reprogrammed in the field. CPLDs are non-volatile; so (once they are programmed with a circuit) the circuit remains implemented in the device even when the power is removed.

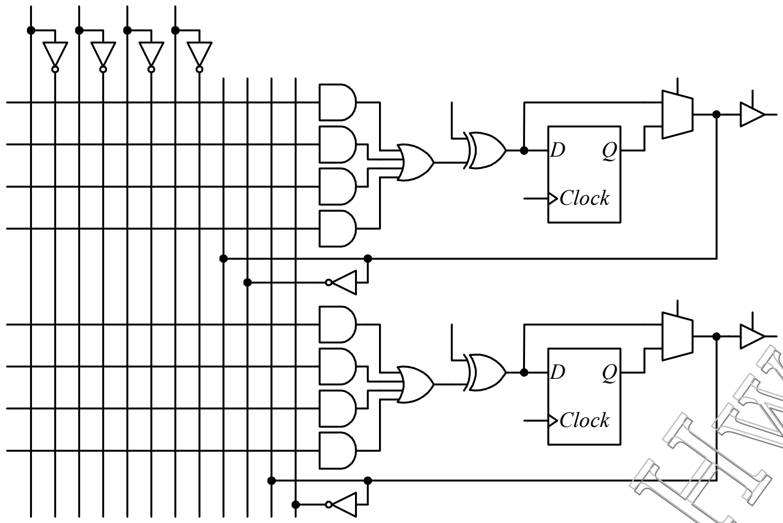


Figure 5.21 Circuit for the logic-array block with two macrocells.

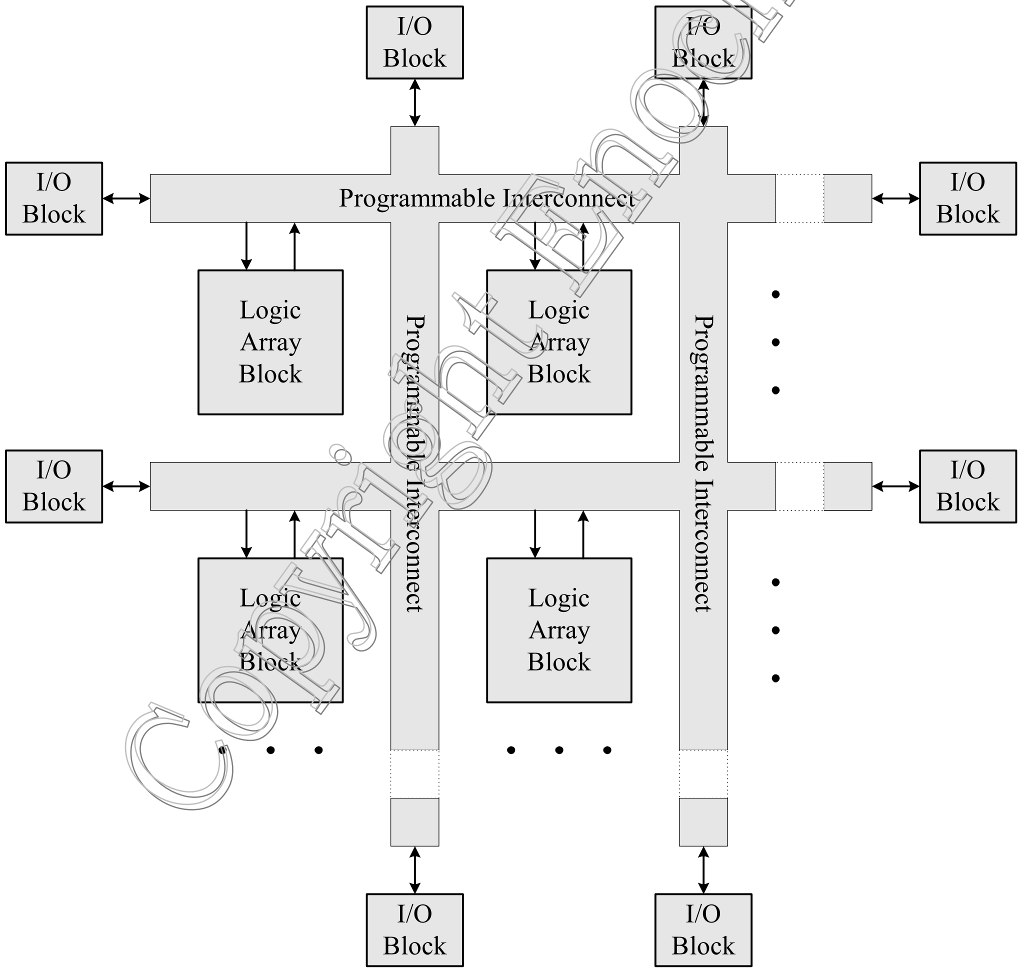


Figure 5.22 Internal circuit for a complex programmable logic device (CPLD).

Feature	MAX7000 CPLD	FLEX10K FPGA
Usable logic gates	10,000	250,000
Macrocells	512	N/A
Logic array blocks	32	1,520
User I/O pins	212	470

Figure 5.23. Features of the Altera MAX7000 CPLD and the FLEX10K250 FPGA.

*5.10 * Field Programmable Gate Array (FPGA)*

Field programmable gate arrays (FPGAs) are complex programmable logic devices that are capable of implementing up to 250,000 logic gates and up to 40,960 RAM bits, as featured by the Altera FLEX10K250 FPGA chip (see Figure 5.23). The internal circuitry of the FLEX10K FPGA is shown in Figure 5.24. The device contains an embedded array and a logic array. The embedded array is used to implement memory functions and complex logic functions, such as microcontroller and digital-signal processing. The logic array is used to implement general logic, such as counters, arithmetic logic units, and state machines.

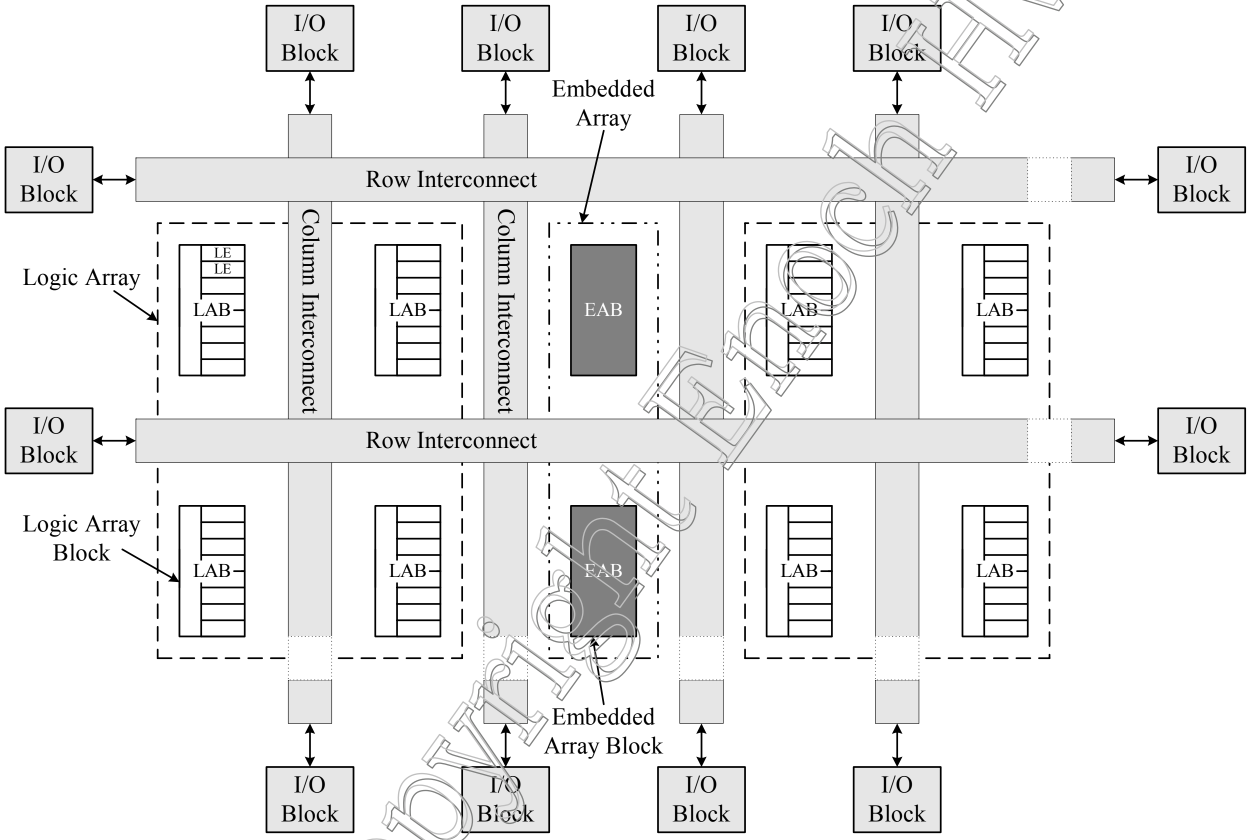


Figure 5.24. FLEX10K FPGA circuit.

The embedded array consists of a series of embedded array blocks (EABs). When implementing memory functions, each EAB provides 2,048 bits, which can be used to create RAM, dual-port RAM, or ROM. EABs can be used independently, or multiple EABs can be combined to implement larger functions.

The logic array consists of multiple logic array blocks (LABs). Each LAB contains eight logic elements (LE) and a local interconnect. The LE shown in Figure 5.25 is the smallest logical unit in the FLEX10K architecture. Each LE consists of a 4-input look-up table (LUT) and a programmable flip-flop. The 4-input LUT is a function generator made from a 16-to-1 multiplexer that can quickly compute any function of four variables. (Refer to Section 4.9.1 on how multiplexers are used to implement Boolean functions.) The four input variables are connected to the four select lines of the multiplexer. Depending on the values of these four variables, the value from one of the 16 multiplexer inputs is passed to the output. There are 16 1-bit registers connected to the 16 multiplexer inputs to supply the multiplexer input values. Depending on the function to be implemented, the content of the 1-bit registers is set to a 0 or a 1. It is set to a 1 for all the 1-minterms of a 4-variable function, and to a 0 for all the 0-minterms. The LUT in Figure 5.25 implements the 4-variable function $F(w,x,y,z) = \Sigma(0, 3, 5, 6, 7, 12, 13, 15)$. The programmable flip-flop can be configured for D, T, JK, or SR operations, and is used for sequential circuits. For combinational circuits, the flip-flop can be bypassed using the 2-to-1 multiplexer.

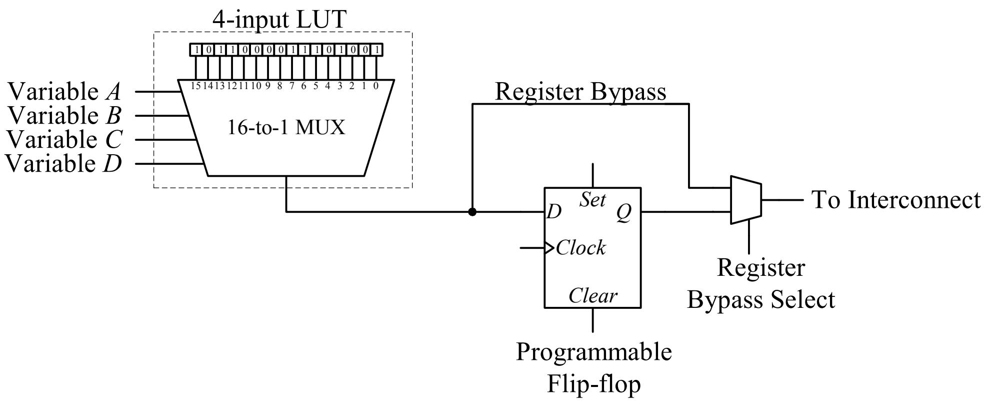


Figure 5.25. Logic element circuit with a 4-input LUT and a programmable register.

All of the EABs, LABs, and I/O elements, are connected together via the FastTrack interconnect, which is a series of fast row and column buses that run the entire length and width of the device. The interconnect contains programmable switches so that the output of any block can be connected to the input of any other block.

Each I/O pin in an I/O element is connected to the end of each row and column of the interconnect and can be used as either an input, output, or bi-directional port.

5.11 Summary Checklist

- ❑ Voltage levels
- ❑ Weak 0, weak 1
- ❑ NMOS
- ❑ NMOS truth table
- ❑ PMOS
- ❑ PMOS truth table
- ❑ High-impedance *Z*
- ❑ Transistor circuits for basic gates
- ❑ PLD
- ❑ ROM circuit implementation
- ❑ PLA circuit implementation
- ❑ PAL circuit implementation
- ❑ CPLD
- ❑ FPGA

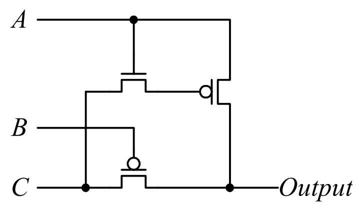
5.12 Problems

5.1 Draw the CMOS circuit for a 2-input NOR gate.
5.2 Draw the CMOS circuit for a 2-input OR gate.
5.3 Draw the CMOS circuit for a 3-input NAND gate.
5.4 Draw the CMOS circuit for a 3-input NOR gate.
5.5 Draw the CMOS circuit for an AND gate by using two NMOS transistors for the 0 half of the circuit and two PMOS transistors for the 1 half of the circuit.

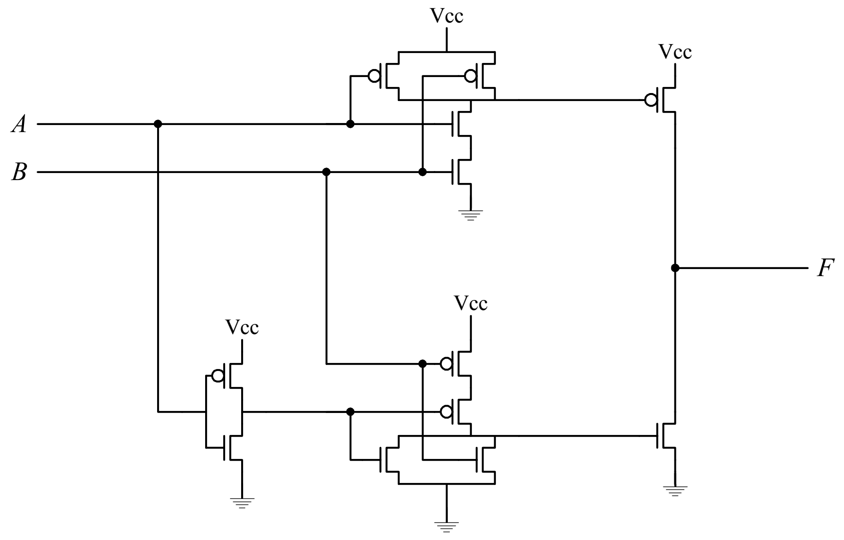
5.6 Draw the CMOS circuit for a 3-input AND gate.

5.7 Derive the truth table for the following CMOS circuits. There are six possible values: 1, 0, Z, weak 1, weak 0, and short.

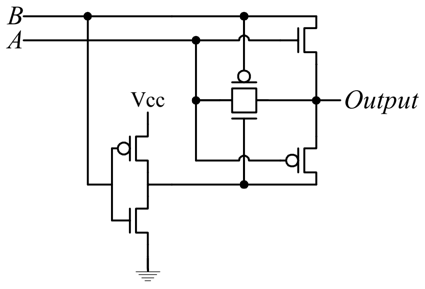
a)



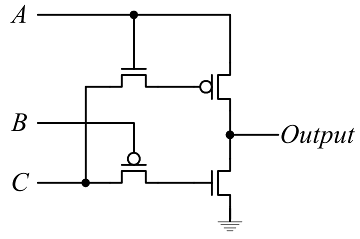
b)



c)



d)



5.8 Synthesize a CMOS circuit that realizes the following truth table.

A	*B*	*Output*
0	0	0
0	1	0
1	0	1
1	1	0

5.9 Synthesize a CMOS circuit that realizes the following truth table having two inputs and one output. Use as few transistors as possible.

A	*B*	*Output*
0	0	0
0	1	0
1	0	1
1	1	Short

5.10 Use one 16 × 4 ROM (4 address lines, 16 entries, 4 data lines) to implement the following functions. Label all of the lines clearly.

$f_1 = w'xy'z + w'xz$
$f_2 = w$
$f_3 = xy' + xyz$

5.11 Use one 16 × 4 ROM (4 address lines, 16 entries, 4 data lines) to implement the following functions. Label all of the lines clearly.

$f_1 = w\,x'\,y'\,z + w\,x'\,y\,z' + w'\,x\,y'\,z'$
$f_2 = x\,y + w'\,z + w\,x'\,y$

5.12 Use one 4 × 8 × 4 PLA to implement the following two functions:

$F_1(w,x,y,z) = wx'y'z + wx'yz' + wxy'$
$F_2(w,x,y,z) = wx'y + x'y'z$

5.13 Use one 4 × 8 × 4 PLA to implement the following two functions:

$F_1(w,x,y,z) = \Sigma(0,2,3,4,5,6,11,12,13,14,15)$
$F_2(w,x,y,z) = \Sigma(1,2,3,5,7,9)$

Index